Homework 4

Q-learning Algorithms

I would like the last of my extra-late days for this assignment

Rem: I had many weird issues when running my experiment. It seems that not all the random usage in the code has been correctly seeded with the seed given in the configuration file. My version of the packages installed is not strictly the same as the one required, to be able to run the experiment on my pc. I encountered many weird problems with my algorithm not achieving the expected results, but when I strictly use the same code and same commands on a classmate's computer, my code is reaching the expected values. I decided so to use the results with the expectation that you will understand my situation.

Question 1

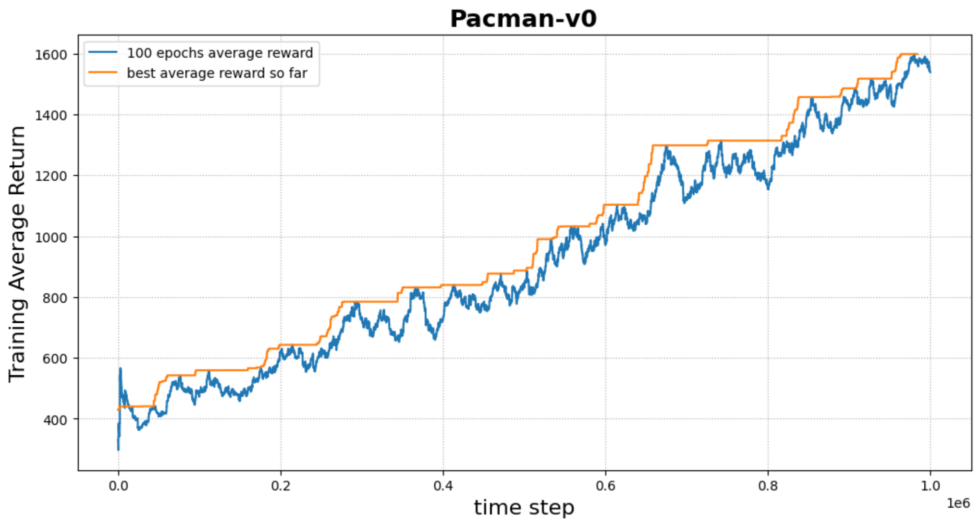


Figure 1: Training average return (100 epochs) and Best average reward so far on the Pac-Man Atari environment. Experiment based on the edited configuration file as shown in **Appendix**, note also that the number of agent update per iteration was set to 1.

Q1 commands

```
python run_hw4.py env_name=MsPacman-v0 exp_name=q1 n_iter=1000000 \
num_agent_train_steps_per_iter=1
```

Question 2

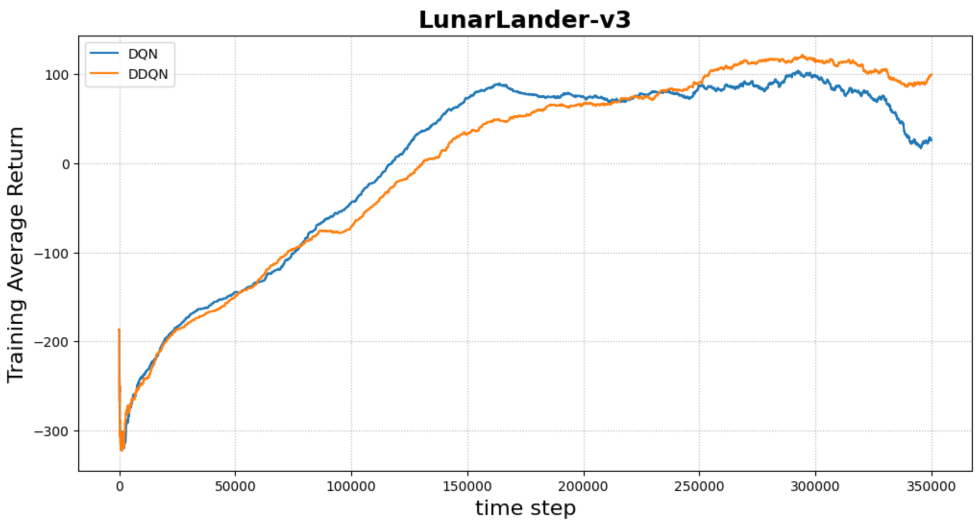


Figure 2: Training average return of DQN and DDQN agent on LunarLander environment. Note that for each learning curve, 3 runs were averaged. Those individually runs are plotted at Fig.3. Some may notice that after its peak, DQN's performance decrease as time step goes by. That is not the case of DDQN, which smoothly increase its performance as expected because it should have less bias estimates of q-values. Note also that the number of agent update per iteration was set to 1.

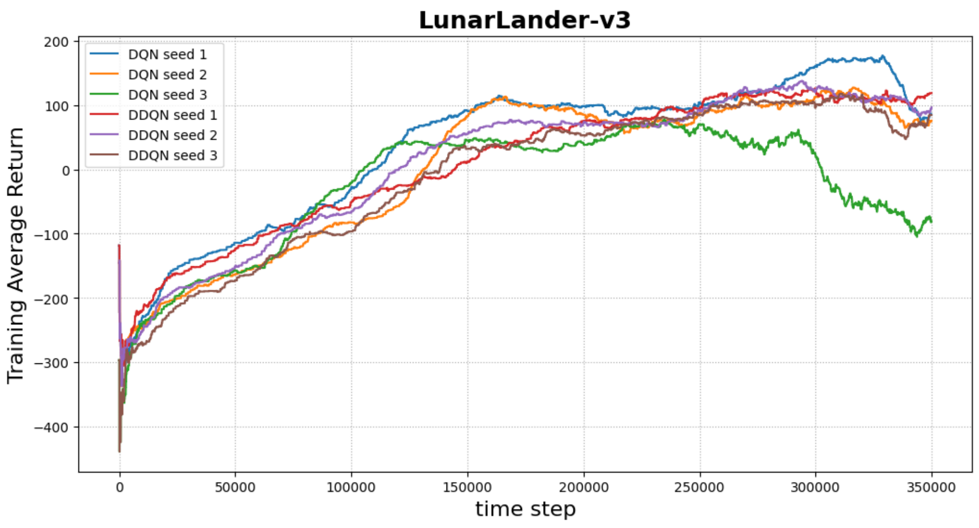


Figure 3: Training average return of different seeded DQN and DDQN agent runs on LunarLander environment. Note also that the number of agent update per iteration was set to 1.

Q2 commands

```
python3 run_hw4.py env_name=LunarLander-v3 exp_name=q2_dqn_1 seed=1 n_iter=350000 \
num_agent_train_steps_per_iter=1

python3 run_hw4.py env_name=LunarLander-v3 exp_name=q2_dqn_2 seed=2 n_iter=350000 \
num_agent_train_steps_per_iter=1

python3 run_hw4.py env_name=LunarLander-v3 exp_name=q2_dqn_3 seed=3 n_iter=350000 \
num_agent_train_steps_per_iter=1

python3 run_hw4.py env_name=LunarLander-v3 exp_name=q2_doubledqn_1 double_q=True \
seed=1 n_iter=350000 num_agent_train_steps_per_iter=1

python3 run_hw4.py env_name=LunarLander-v3 exp_name=q2_doubledqn_2 double_q=True \
seed=2 n_iter=350000 num_agent_train_steps_per_iter=1

python3 run_hw4.py env_name=LunarLander-v3 exp_name=q2_doubledqn_3 double_q=True \
seed=3 n_iter=350000 num_agent_train_steps_per_iter=1
```

Question 3

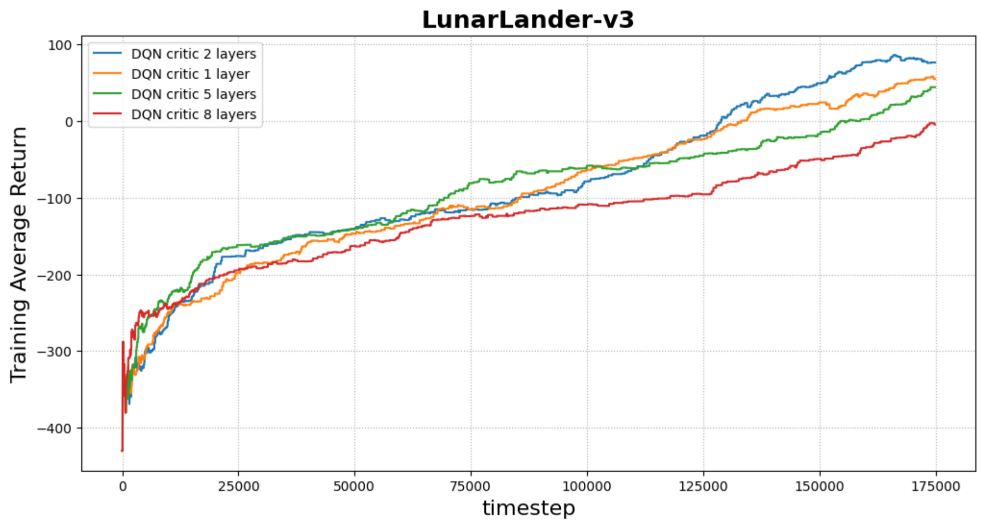


Figure 4: Training average return of DQN agents on LunarLander environment. Each curve is a different critic architecture of the DQN agent, going from 1 layers to 8. Unsurprisingly, we can see that the best architecture is to use 2 layers for the critic. The next best result is the critic using 1 layer, which have probably less good result due to a less capacity. Then we have the critic which use 5 layers that has been slightly better between the 750000 and the 100000 time steps and the got performance under the critic that use 1 layer which can be due to a lack of generalization and maybe not enough training. Finally, we have the critic composed of 8 layers which has the worst performance, which is clearly due to a lack of generalization and to short training. Note also that the number of agent update per iteration was set to 1.

Q3 commands

```
python run_hw4.py env_name=LunarLander-v3 exp_name=q3_default \
n_iter=350000 num_agent_train_steps_per_iter=1

python run_hw4.py env_name=LunarLander-v3 exp_name=q3_layer1 \
n_iter=350000 num_agent_train_steps_per_iter=1 n_layers_critic=1

python run_hw4.py env_name=LunarLander-v3 exp_name=q3_layer5 \
n_iter=350000 num_agent_train_steps_per_iter=1 n_layers_critic=5

python run_hw4.py env_name=LunarLander-v3 exp_name=q3_layer8 \
n_iter=350000 num_agent_train_steps_per_iter=1 n_layers_critic=8
```

Question 4

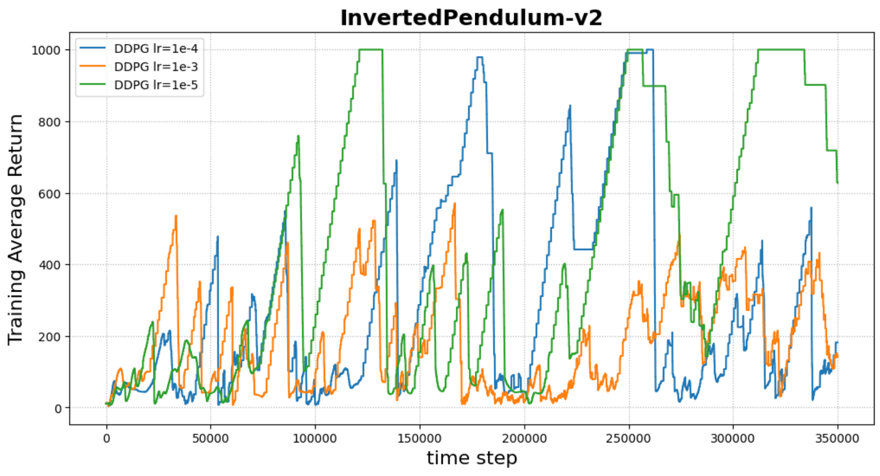


Figure 5: Training average return (100 epochs) of DDPG agents trained using different learning rates on the InvertedPendulum environment. The learning rate chosen for the training was respectively 1e-3 (default), 1e-4 and 1e-5. By looking at the curves, we can clearly notice a correlation between the performance and the learning rate. As low learning rate as 1e-5 leads to the best results with more stability, followed by 1e-4 which timidly achieved 1000 average reward for a short time. Finally, 1e-3 which not solved the task and reach a maximum average reward of 600. Note that the noise used for trained those agents was sampled from a standard normal distribution ($\mathcal{N}(0, 1)$) rescaled by a factor of 0.1. Also the number of agent and critic update per iteration were set to 1.

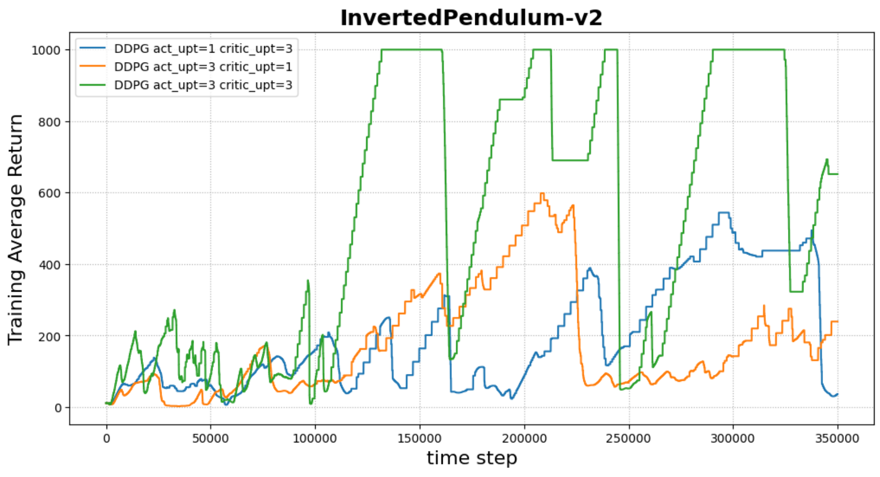


Figure 6: Training average return (100 epochs) of DDPG agents trained using different update strategies on the InvertedPendulum environment. The strategies consisted of testing how much updating the critic influences DDPG, then try with updating the actor more often to finally updates both critic and target more often. As showed by the curves, it seems important for DDPG to update more often than 1 time per iteration its actor and its critic. Note that the noise used for trained those agents was sampled from a standard normal distribution ($\mathcal{N}(0, 1)$) rescaled by a factor of 0.1. The learning rate was fixed to 1e-5 for all experiments as it was the best as shown by Fig.5.

TODO: RENAME THE RUN

Q4 commands

```
python3 run_hw4.py exp_name=q4_ddpg_up1_1_lr1e-4 rl_alg=ddpg \
env_name=InvertedPendulum-v2 atari=False learning_freq=1 \
num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 \
n_iter=350000 learning_rate=1e-4

python3 run_hw4.py exp_name=q4_ddpg_up1_1_lr1e-3rl_alg=ddpg \
env_name=InvertedPendulum-v2 atari=False learning_freq=1 \
num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 \
n_iter=350000 learning_rate=1e-3

python3 run_hw4.py exp_name=q4_ddpg_up1_1_lr1e-5 rl_alg=ddpg \
env_name=InvertedPendulum-v2 atari=False learning_freq=1 \
num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 \
n_iter=350000 learning_rate=1e-5

python3 run_hw4.py exp_name=q4_ddpg_up1_3_lr1e-5 rl_alg=ddpg \
env_name=InvertedPendulum-v2 atari=False learning_freq=1 \
num_agent_train_steps_per_iter=3 \
num_critic_updates_per_agent_update=1 n_iter=116667 learning_rate=1e-5

python3 run_hw4.py exp_name=q4_ddpg_up3_1_lr1e-5  rl_alg=ddpg \
env_name=InvertedPendulum-v2 atari=False learning_freq=1 \
num_agent_train_steps_per_iter=1 \
num_critic_updates_per_agent_update=3 n_iter=350000 learning_rate=1e-5

python3 run_hw4.py exp_name=q4_ddpg_up3_3_lr1e-5 rl_alg=ddpg \
env_name=InvertedPendulum-v2 atari=False learning_freq=1 \
num_agent_train_steps_per_iter=3 \
num_critic_updates_per_agent_update=3 n_iter=116667 learning_rate=1e-5
```

Question 5

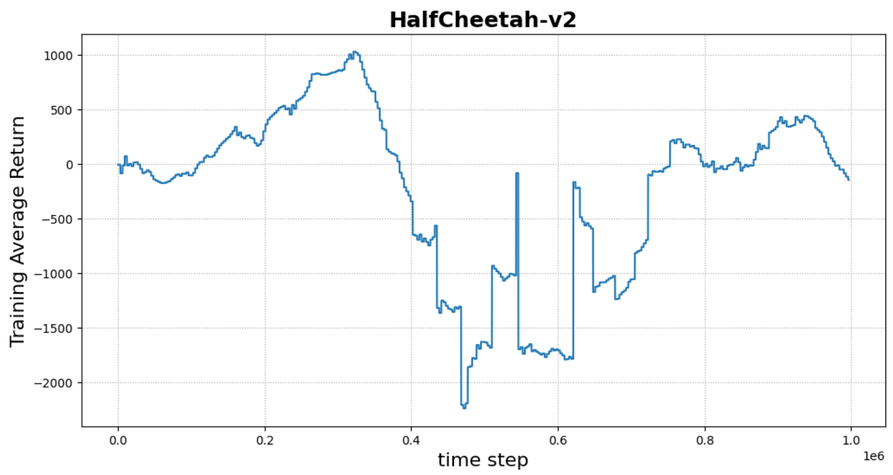


Figure 7: Training average return (100 epochs) of DDPG agent trained on the Half-Cheetah environment using the best hyperparameters from experiment 4. As showed by the curve, the agent struggle to maintain its performance over the time step, which is probably caused by a wrong usage of hyperparameters. Some may found it logical as Half-Cheetah is a very different and more complex environment than InvertedPendulum. Note that the noise used for trained this agent was sampled from a standard normal distribution ($\mathcal{N}(0,1)$) rescaled by a factor of 0.1. The action learning rate was fixed to 1e-5 as it was the best as shown by Fig.5 and the critic one is set to 1e-4. The architecture used for the critic is 2 layers and 64 units as shown in Fig.6. Both agent and critic update per iteration was set to 3.

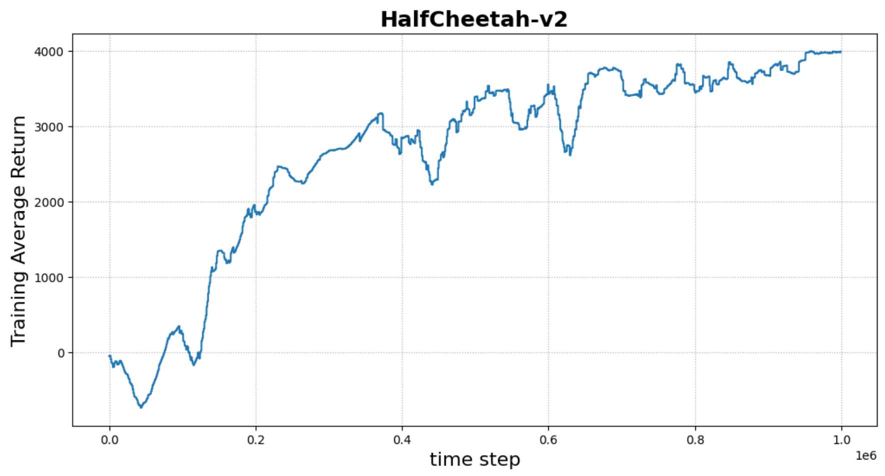


Figure 8: Training average return (100 epochs) of DDPG agent trained on the Half-Cheetah environment. This run is not the one using the best hyperparameters from experiment 4. I have run another experiment with different hyperparameters, as the run using the best ones from q4 did not convince me. It was indeed a good choice as the agent reached 4000 rewards. Note that the noise used for trained this agent was sampled from a standard normal distribution ($\mathcal{N}(0,1)$) rescaled by a factor of 0.1. The action learning rate was fixed to 1e-5 as it was the best as shown by Fig.5 and the critic one is set to 1e-4. Both agent and critic update per iteration was set to 1.

Q5 commands

Run with the best hyperparameters from experiment 4:

```
python run_hw4.py exp_name=q5_ddpg_hard_up3_3_lr1-e5 rl_alg=ddpg \
env_name=HalfCheetah-v2 atari=false learning_freq=1 \
num_agent_train_steps_per_iter=3 num_critic_updates_per_agent_update=3 \
n_iter=333333 learning_rate=1e-5
```

Second run with the tuned hyperparameters:

```
python3 run_hw4.py exp_name=q5_ddpg_hard_up_lr1-e5 rl_alg=ddpg \
env_name=HalfCheetah-v2 atari=false learning_freq=1 \
num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 \
n_iter=1000000 learning_rate=1e-5 critic_learning_rate=1e-4
```

Question 6

Q6 commands

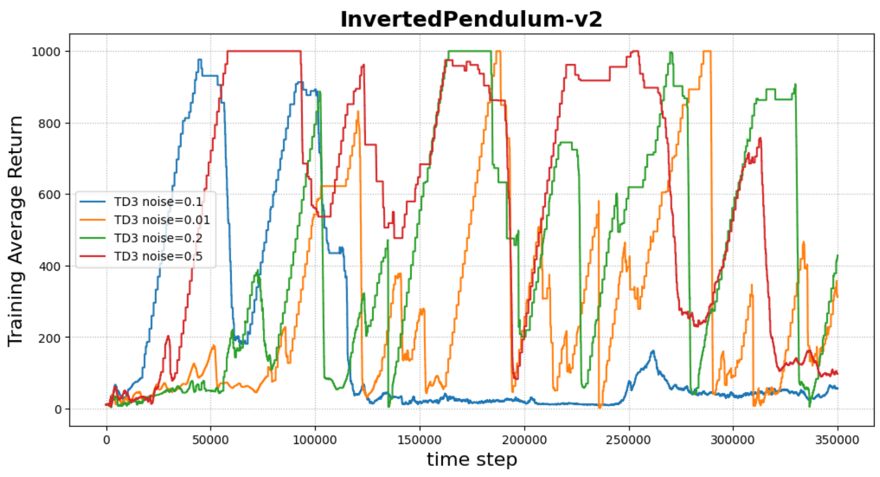


Figure 9: Training average return (100 epochs) of TD3 agent trained using different noise on InvertedPendulum environment.Note that the noise used for trained those agents was sampled from a standard normal distribution ($\mathcal{N}(0, 1)$) rescaled by a factor of, corresponding to the `'noise='` on the legend. As show by the curves, using a noise of 0.5 seems leading to the best result and for a larger time. Strangely, the noise of 0.1 leads to the poorest results. The actor learning rate was fixed to 1e-5 for all experiments as it was the best as shown by Fig.5.

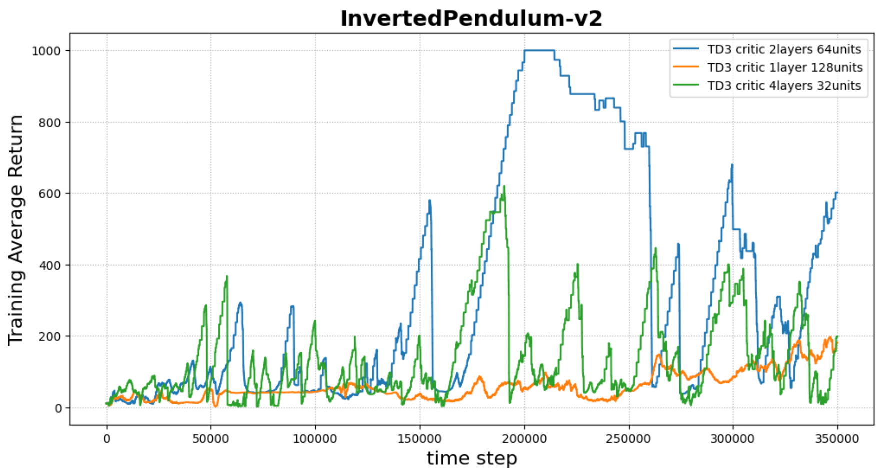


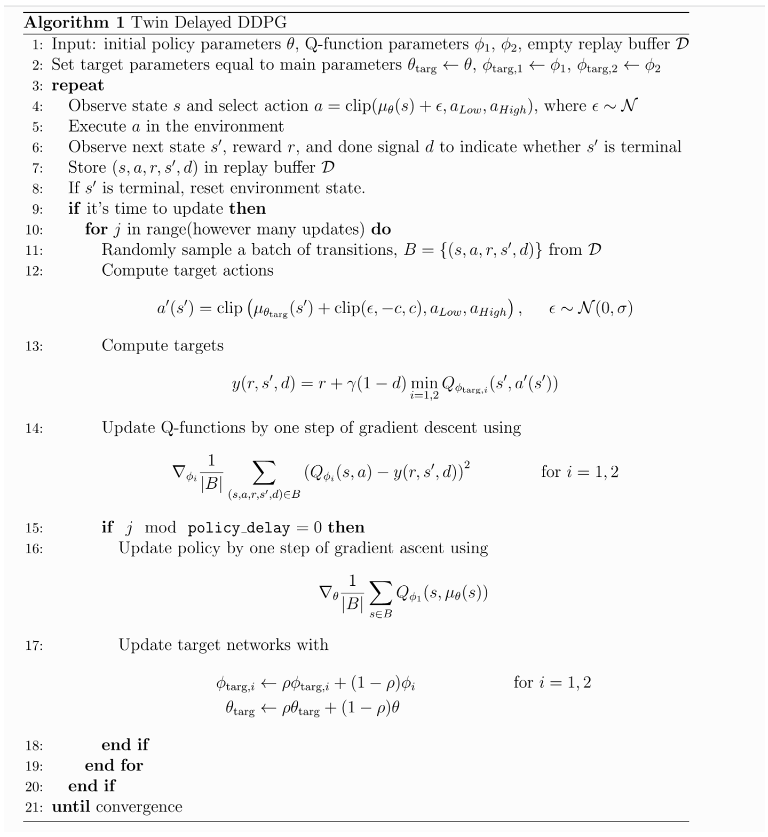
Figure 10: Training average return (100 epochs) of TD3 agent trained using different critic architectures on InvertedPendulum environment. The strategies consisted of always having 128 neurons on each architecture, but dispatched into different layers. Results tend to show that using 2 layers of 64 units is the best architectures for the critic on that environment, with probably less capacity for the critic using 1 layer of 128 neurons and too much for only, 350000 time steps for the one using 4 layers of 32 neurons. Note that the noise used for trained those agents was sampled from a standard normal distribution ($\mathcal{N}(0, 1)$) rescaled by a factor of 0.2. The actor learning rate was fixed to 1e-5 for all experiments as it was the best as shown by Fig.5.

Q6 commands

```
python run_hw4.py exp_name=q6_td3_shape_default_rho_default \
rl_alg=ddpg env_name=InvertedPendulum-v2 atari=false learning_freq=1 \
num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 \
n_iter=350000

 python run_hw4.py exp_name=q6_td3_shape_default_rho_0.01 rl_alg=ddpg \
 env_name=InvertedPendulum-v2 atari=false learning_freq=1 \
 num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 \
 n_iter=350000 td3_target_policy_noise=0.01

 python run_hw4.py exp_name=q6_td3_shape2l128hu_rho0.2 rl_alg=ddpg \
 env_name=InvertedPendulum-v2 atari=false learning_freq=1 \
 num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 \
 n_iter=350000 td3_target_policy_noise=0.2

 python run_hw4.py exp_name=q6_td3_shape_default_rho0.5 rl_alg=ddpg \
 env_name=InvertedPendulum-v2 atari=false learning_freq=1 \
 num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 \
 n_iter=350000 td3_target_policy_noise=0.5

 python run_hw4.py exp_name=q6_td3_shape_2l_64u_rho0.2 rl_alg=ddpg \
 env_name=InvertedPendulum-v2 atari=false learning_freq=1 \
 num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 \
 n_iter=350000 td3_target_policy_noise=0.2 n_layers_critic=2 size_hidden_critic=64

 python run_hw4.py exp_name=q6_td3_shape_1l_128u_rho0.2 rl_alg=ddpg \
 env_name=InvertedPendulum-v2 atari=false learning_freq=1 \
 num_agent_train_steps_per_iter=1\
 num_critic_updates_per_agent_update=1 n_iter=350000 td3_target_policy_noise=0.2 \
 n_layers_critic=1 size_hidden_critic=128

 python run_hw4.py exp_name=q6_td3_shape_4l_32u_rho0.2 rl_alg=ddpg \
 env_name=InvertedPendulum-v2 atari=false learning_freq=1 \
 num_agent_train_steps_per_iter=1 num_critic_updates_per_agent_update=1 n_iter=350000 \
 td3_target_policy_noise=0.2 n_layers_critic=4 size_hidden_critic=32
```

BONUS: Question 6 (TD3 with 2 critics)

As a bonus we were asked to implement the real version of TD3 using two critics which I did. I took inspiration from the OpenAI blogpost named ”SpinningUp” which provide the algorithm as shown in Fig.11



Algorithm 1 Twin Delayed DDPG

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
3: repeat
4:   Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5:   Execute $a$ in the environment
6:   Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:   Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:   If $s'$ is terminal, reset environment state.
9:   if it's time to update then
10:    for $j$ in range(however many updates) do
11:      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:      Compute target actions
             $a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$
13:      Compute targets
             $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$
14:      Update Q-functions by one step of gradient descent using
             $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \quad \text{for } i = 1, 2$
15:      if $j$ mod policy_delay $= 0$ then
16:        Update policy by one step of gradient ascent using
               $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$
17:        Update target networks with
               $\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \quad \text{for } i = 1, 2$
               $\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$
18:      end if
19:    end for
20:  end if
21: until convergence
```

Figure 11: SpinningUp's version of TD3 algorithm.

The code is available in the class `td3agent.py` and `td3critic.py`. The algorithm was also used to trained the agent to answered the question 7.

Question 7

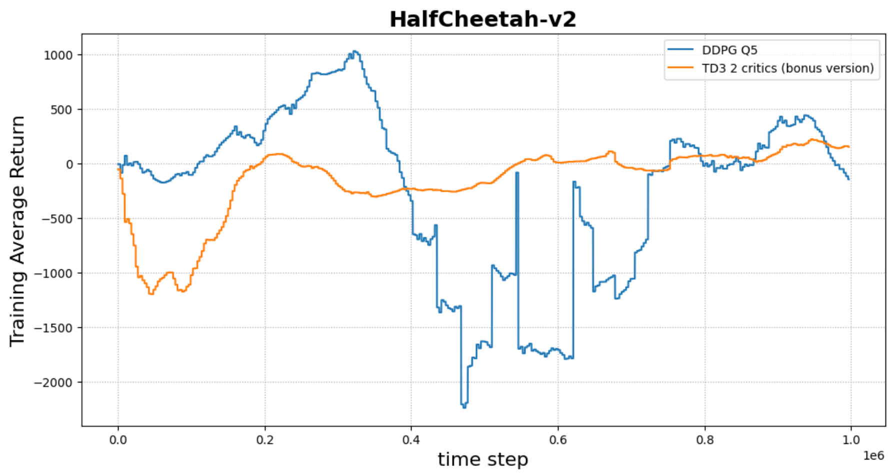


Figure 12: Training average return (100 epochs) of DDPG agent from question 5 versus the bonus version of TD3 on Half-Cheetah environment Some may notice that those result were not those that we should expect for such an experiment. This can be the result of different reasons. Firstly, those algorithms could be wrongly fine-tuned, as Half-Cheetah is a very different and more complex environment than InvertedPendulum. Secondly, it is not excluded that the algorithm just doesn't learn properly due to an error in the implementation. Finally, those algorithms as not the most easy to make work as they are noise dependent and have a lot of different possible hyperparameters to tune. All the information about the DDPG agent can be found in the caption of Fig.??, for TD3 part you may want to have a look at the command just below.

Q7 commands

```
python run_hw4.py exp_name=q7_td3_shape_default_u_3_3_rho0.5 rl_alg=td3 \
env_name=HalfCheetah-v2 atari=false  num_agent_train_steps_per_iter=3 \
num_critic_updates_per_agent_update=3  learning_freq=1 n_iter=333333 \
td3_target_policy_noise=0.5 learning_rate=1e-5 critic_learning_rate=1e-4
```

Appendix

Default configuration file used to run the experiments. Note that I add some properties to configure TD3.

```
env_name: 'LunarLander-v3'
atari: True
ep_len: 200
exp_name: 'todo'
double_q: False
batch_size: 64
train_batch_size: 64
eval_batch_size: 4096
num_agent_train_steps_per_iter: 2
num_critic_updates_per_agent_update: 2
seed: 1
```

```
no_gpu: False
which_gpu: 0
video_log_freq: -1
scalar_log_freq: 1
save_params: False
rl_alg: 'dqn'
learning_starts: 1024
learning_freq: 0
target_update_freq: 1
exploration_schedule: 0
optimizer_spec:  0
replay_buffer_size: 1000000
frame_history_len: 1
gamma: 0.99
n_layers_critic: 2
size_hidden_critic: 64
critic_learning_rate: 1e-3
n_layers: 2
size: 64
learning_rate: 1e-3
ob_dim: 0
ac_dim: 0
discrete: True
grad_norm_clipping: True
n_iter: 10000
polyak_avg: 0.01 ##
# ==================== #
# TD3 CONFIG PARAMETERS #
# ==================== #
td3_target_policy_noise: 0.1 ##
td3_noise_clip: 0.5
td3_two_q_net: False
td3_policy_delay: 2
```